

COMPARISON OF MATLAB, OCTAVE, AND PYTHON (FOR USE IN THE NUMERICAL ANALYSIS CLASSROOM)

ED BUELER

On the next page are two algorithms each in MATLAB/OCTAVE (left column) and PYTHON (right column). To download these examples, go to bueler.github.io/M615S17 and follow links in the left-hand column of the screen.

A bit of background is useful. MATLAB ([mathworks.com](https://www.mathworks.com)) was designed by Cleve Moler before 1980 for teaching numerical linear algebra without needing FORTRAN. It has since become a powerful programming language and engineering tool. More than half of UAF 600-level math/science/engineering students are already familiar with it. It is available in most labs and graduate student offices at UAF. It works well, looks good, and I like it.

But I do prefer free, open source software. There are effective, free alternatives to MATLAB, and at least two of these work well for this course. First, OCTAVE is a MATLAB clone. The “.m” examples on the next page work in an identical way in MATLAB and in OCTAVE. I will mostly use OCTAVE myself for teaching, but I’ll test examples in both OCTAVE and MATLAB. To download OCTAVE, go to www.gnu.org/software/octave.

Second is the general-purpose language PYTHON (python.org). With the SCIPY (scipy.org) and MATPLOTLIB (matplotlib.org) libraries it has all of MATLAB functionality plus quite a bit more. Using it with the IPYTHON interactive shell (ipython.org) gives the most MATLAB-like experience; this combination is called PYLAB. Students who already use PYTHON will like this option.

Here are some brief “how-to” comments for the MATLAB/OCTAVE examples: `gaussint.m` is a *script*. A script is run by starting MATLAB/OCTAVE, usually in the directory containing the script you want to run. Then type the name of the script at the prompt, without the “.m”:

```
>> gaussint
```

Typing

```
>> help gaussint
```

shows the block of comments as documentation. The second algorithm `bis.m` is a *function* which needs inputs. At the prompt enter

```
>> f = @(x) cos(x) - x
>> bis(0,1,f)
```

for example. We have given `bis.m` three arguments; the last is an “anonymous function.”

For the PYTHON versions: Run PYTHON or IPYTHON to get a prompt. Type `run gaussint.py`. (You can also do `python gaussint.py` directly from a shell.) For the function `bis.py`, first do `from bis import bis`. Then, in IPYTHON, type `bis?` to get documentation, i.e. the “docstring.” At the plain PYTHON prompt, print the docstring directly by `print bis.func_doc`. Run the example as shown in the docstring.

`gaussint.m`

```
% plot the integrand and approximate
% the integral
% / 1
% | exp(-x^2/pi) dx
% / 0
% by left-hand, right-hand, and
% trapezoid rules

N = 1000;
dx = (1 - 0) / N;
x = linspace(0,1,N+1);
y = exp(- x.^2 / pi);

plot(x,y)
axis([0 1 0 1]), grid

format long
lhand = dx * sum(y(1:end-1))
rhand = dx * sum(y(2:end))
trap = (dx/2) * sum(y(1:end-1)+y(2:end))
exact = (pi/2) * erf(1/sqrt(pi))
```

`gaussint.py`

```
#!/usr/bin/env python

# plot the integrand and approximate
# the integral
# / 1
# | exp(-x^2/pi) dx
# / 0
# by left-hand, right-hand, and
# trapezoid rules

from pylab import plot,axis,linspace,sum, \
pi,sqrt,exp,show,grid
from scipy.special import erf

N = 1000
dx = (1.0 - 0.0) / N
x = linspace(0.0,1.0,N+1)
y = exp(- x**2 / pi)

plot(x,y)
axis([0.0,1.0,0.0,1.0]); grid(True)

lhand = dx * sum(y[:-1])
print "lhand = %.15f" % lhand
rhand = dx * sum(y[1:])
print "rhand = %.15f" % rhand
trap = (dx/2) * sum(y[:-1]+y[1:])
print "trap = %.15f" % trap
exact = (pi/2) * erf(1/sqrt(pi))
print "exact = %.15f" % exact
show()
```

`bis.m`

```
function c = bis(a,b,f)
% BIS Apply the bisection method to solve
% f(x) = 0
% with initial bracket [a,b]. Example:
% >> f = @(x) cos(x) - x % define fcn
% >> r = bis(0,1,f) % find root
% >> f(r) % confirm

if (feval(f,a)) * (feval(f,b)) > 0
error('not a bracket!'), end
for k = 1:100
c = (a+b)/2;
r = feval(f,c);
if abs(r) < 1e-12
return % we are done
elseif feval(f,a) * r >= 0.0
a = c;
else
b = c;
end
end
error('no convergence')
```

`bis.py`

```
def bis(a,b,f):
""" BIS Apply the bisection method to solve
f(x) = 0
with initial bracket [a,b]. Example:

from bis import bis
def f(x):
from math import cos
return cos(x) - x
r = bis(0.0,1.0,f)
print([r,f(r)]) """

if f(a) * f(b) > 0.0:
print "not a bracket!"
return
for k in range(100):
c = (a+b)/2
r = f(c)
if abs(r) < 1e-12:
return c # we are done
elif f(a) * r >= 0.0:
a = c
else:
b = c
print "no convergence"
return
```